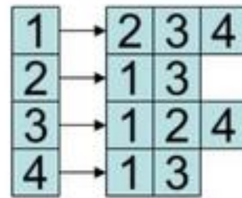
خوارزمية البحث في العرض Breadth First Search BFS

ملاحظة: يعتمد البحث في BFS على مبدأ الرتل queue

ملاحظة: هذه الخوارزمية هي للبيان الموجه، أي أن الرابط (1, 2) يختلف عن الرابط (2, 1).

مبدأ العمل:

- سننشئ مصفوفة; كل عنصر من المصفوفة هو list. تحوي كل list على العقد المجاورة للعنصر مباشرة، وذلك بهدف تمثيل البيان بقائمة تجاور تشبه الشكل التالي مثلاً:



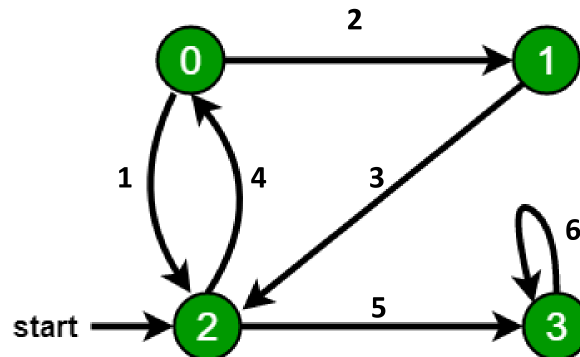
- نستطيع معرفة هل عقدة ما مزارة أم لا بواسطة مصفوفة اسمها visited من نوع Boolean وجميع العقد في البداية غير مزارة.
- ننشئ رتل
- عقدة البداية s بالتأكيد غير مزارة، لذا سنحولها إلى مزارة
- ثم نضيفها للرتل.
- نفحص الرتل باستمرار: طالما أنه غير فارغ {نسحب عقدة من بدايته ثم نطبعها

ثم نفحص جوارها بواسطة iterator.

طالما كان للعقدة جوار{ نقفز لقعدة الجوار ثم نفحصها هل هي غير مزارة؟ اذا كانت غير مزارة سنحولها إلى مزارة ثم نضيفها للرتل}}

ملاحظة: كما ذكرنا في المحاضرة السابقة: يؤخذ بعين الاعتبار تسلسل انشاء الروابط Edge في التابع main، فالرابطة المنشأة أولاً ستدخل إلى قائمة التجاور Adjacency List أولاً، وبالتالي سيتم العبور والتجوال لها أولاً.

مثال: اكتب كود خوارزمية البحث بالعرض BFS بلغة ++C، ثم أوجد التجوال للبيان التالي بدءً من العقدة 2، مع ملاحظة أن الأرقام الموجودة على الروابط هي لتوضيح تسلسل انشائها.



الحل:

```
// Program to print BFS traversal from a given
// source vertex. BFS(int s) traverses vertices
// reachable from s.
#include<iostream>
#include <list>

using namespace std;

// This class represents a directed graph using
// adjacency list representation
class Graph
{
	int V; // No. of vertices

	// Pointer to an array containing adjacenc lists
	list<int> *adj;
public:
	Graph(int V); // Constructor

	// function to add an edge to graph
	void addEdge(int v, int w);

	// prints BFS traversal from a given source s
	void BFS(int s);
};

Graph::Graph(int y)
{
	V = y;
	adj = new list<int>[V];
}

void Graph::addEdge(int v, int w)
{
	adj[v].push_back(w); // Add w to v’s list.
}
```

```
void Graph::BFS(int s)
{
	// Mark all the vertices as not visited
	bool *visited = new bool[V];
	for(int i = 0; i < V; i++)
		visited[i] = false;

	// Create a queue for BFS
	list<int> queue;

	// Mark the current node as visited and enqueue it
	visited[s] = true;
	queue.push_back(s);

	// 'i' will be used to get all adjacent
	// vertices of a vertex
	list<int>::iterator i;

	while(!queue.empty())
	{
		// Dequeue a vertex from queue and print it
		s = queue.front();
		cout << s << " ";
		queue.pop_front();

		// Get all adjacent vertices of the dequeued
		// vertex s. If a adjacent has not been visited,
		// then mark it visited and enqueue it
		for (i = adj[s].begin(); i != adj[s].end(); ++i)
		{
			if (!visited[*i])
			{
				visited[*i] = true;
				queue.push_back(*i);
			}
		}
	}
}

// Driver program to test methods of graph class
int main()
{
	// Create a graph given in the above diagram
	Graph g(4);
	g.addEdge(0, 2);
	g.addEdge(0, 1);
	g.addEdge(1, 2);
	g.addEdge(2, 0);
	g.addEdge(2, 3);
	g.addEdge(3, 3);

	cout << "Following is Breadth First Traversal "
		<< "(starting from vertex 2) \n";
```

```
	g.BFS(2);
	system("pause");
	return 0;
}
```

وبالتالي يكون التجوال هو: **2 0 3 1**

تمرين: قم بإيجاد التجوال للبيان السابق بدءً من العقدة 3، ماذا تلاحظ؟

الحل: نغيّر في الاستدعاء ليصبح ;g.BFS(3)

نلاحظ أنه سيتم التجوال فقط على العقدة 3 لأن البيان موجه، ولا يوجد أي رابط من 3 إلى باقي العقد، ويوجد حلقة Loop للعقدة 3.